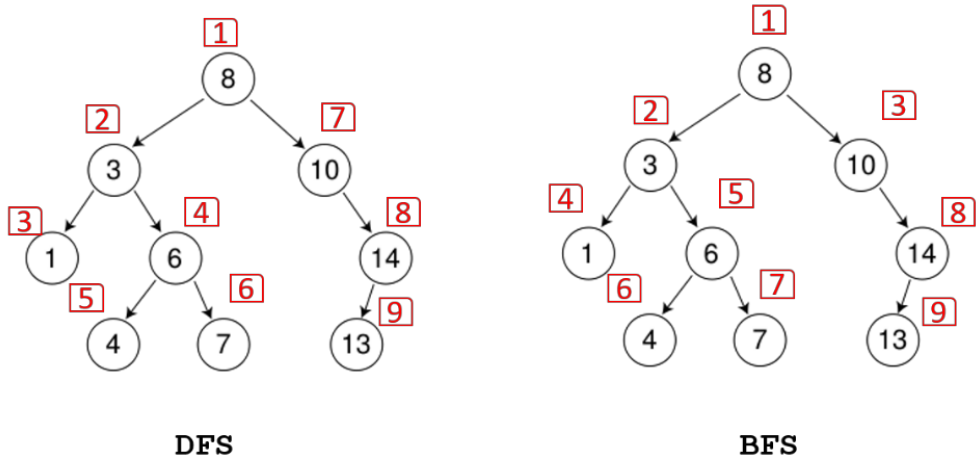
3. Depth First Search

Depth first search (DFS) เป็นขั้นตอนวิธีในการท่อง (traverse) หรือค้นหา (search) บนกราฟ โดยต้องกำหนดโหนดที่เป็นจุดเริ่มต้น จากนั้นขั้นตอนวิธีจะ traverse ไปยังโหนดอื่น ๆ ของกราฟที่อยู่ติดกัน เรียกว่าไป visit โหนด DFS มีลักษณะการ visit โหนดอื่น ๆ โดยต้องการเดินไปข้างหน้าให้<u>ลึกที่สุด</u> จนพบทางตัน จากนั้นจึงค่อยเดินย้อนกลับมา ระหว่างทางที่เดินกลับ หากพบโหนดที่ยังไม่ได้ visit ก็จะเข้าไป visit



รูปที่ 2

DFS สามารถนำมาใช้กับข้อมูลรูปแบบ 2 มิติอย่างอาร์เรย์ 2 มิติได้ เมื่อเราพิจารณาให้ข้อมูลที่แถว r และคอลัมน์ c (เรียกว่า cell) เป็นโหนด จากนั้น DFS จะ traverse ไปยัง cell ที่อยู่ติดกันได้ ซึ่งมีได้มากที่สุด 8 cell จากรูปที่ 3 ถ้า X เป็น cell ที่แทนโหนดปัจจุบัน DFS จะสามารถ traverse ไปยัง A, U, B, L, R, C, D และ E ได้ในกรณี 8 ทิศ หรือ สามารถ traverse ไปยัง U, L, R และ D ได้ในกรณี 4 ทิศ

คอลัมน์ c

	A	U	B
แถว r	L	X	R
	C	D	E

รูปที่ 3

เมื่อกำหนดให้จุดเริ่มต้นเป็น cell ที่แถว 0 และคอลัมน์ 0 เดินได้ 4 ทิศ และ priority ของการเลือกทิศจากมากไปน้อยคือ **U, R, D** และ **L** ลำดับการท่องอาร์เรย์ 2 มิติของ DFS แสดงในรูปที่ 4

(0,0) (0,1) (0,2) (0,3) (1,3) (2,3) (2,2) (1,2) (1,1) (2,1) (2,0) (1,0)

	0	1	2	3
0	1	2	3	4
1	12	9	8	5
2	11	10	7	6

รูปที่ 4

- เริ่มที่ (0,0) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ จากนั้นจะเลือกเดินในทิศ R ซึ่งเดินได้ไปยัง (0,1)
- จาก (0,1) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ จากนั้นจะเลือกเดินในทิศ R ซึ่งเดินได้ไปยัง (0,2)
- จาก (0,2) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ จากนั้นจะเลือกเดินในทิศ R ซึ่งเดินได้ไปยัง (0,3)
- จาก (0,3) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ จากนั้นจะเลือกเดินในทิศ D ซึ่งเดินได้ไปยัง (1,3)
- จาก (1,3) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ จากนั้นจะเลือกเดินในทิศ D ซึ่งเดินได้ไปยัง (2,3)
- จาก (2,3) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ ซึ่งเดินได้ไปยัง (2,2)
- จาก (2,2) DFS จะเลือกเดินไปในทิศ U ซึ่งเดินได้ไปยัง (1,2)
- จาก (1,2) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ D แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ L ซึ่งเดินได้ไปยัง (1,1)
- จาก (1,1) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ D ซึ่งเดินได้ไปยัง (2,1)
- จาก (2,1) DFS จะเลือกเดินไปในทิศ U แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ R แต่เดินไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ D แต่เดินไปไม่ได้ (ซ้ำ) จากนั้นจะเลือกเดินในทิศ L ซึ่งเดินได้ไปยัง (2,0)
- จาก (2,1) DFS จะเลือกเดินไปในทิศ U ซึ่งเดินได้ไปยัง (1,0)

- จาก (1,0) ไม่มีทิศใดที่เดินได้แล้ว จบการทำงาน

หากเปลี่ยน priority ของการเลือกทิศจากมากไปน้อยเป็น L, R, U และ D ลำดับของการเดินจะเป็นดังรูป 5 แทน

	0	1	2	3
0	1	2	3	4
1	8	7	6	5
2	9	10	11	12

รูปที่ 5

เพื่อป้องกันการ vitsit ซ้ำ ขั้นตอนวิธี DFS จะเก็บอาร์เรย์ 2 มิติเพื่อใช้บันทึกว่า cell ใดบ้างที่เคย visit มาแล้ว

โปรแกรมที่ 2.4 การทำงานของขั้นตอนวิธีของ DFS แบบ recursive บนอาร์เรย์ 2 มิติ

```
#include<iostream>
#define MAX 1005
using namespace std;
// Globals
int M;                  // number of rows
int N;                  // number of columns
bool visit[MAX][MAX];   // keeps tracks of visited cells
// Forwards
void recursiveDFS(int r, int c);
bool isValid(int r, int c);
int main(){
    ios_base::sync_with_stdio(false);   // avoid syn C++ streams
    cin.tie(NULL);                      // flood cout before cin

    cin >> M >> N;
    cout << "Recursive DFS\n";
    recursiveDFS(0,0);
    cout << endl;
    return 0;
}
void recursiveDFS(int r, int c){
    visit[r][c] = true;
    cout << "(" << r << "," << c << ") ";


```

```
    // order of priority
    // up, right, down, left
    int dx[] = {-1,0,1,0};
    int dy[] = {0,1,0,-1};
    for(int i=0; i<4; i++)
        if(isValid(r + dx[i], c + dy[i]))
            recursiveDFS(r + dx[i], c + dy[i]);
}
bool isValid(int r, int c){
    return (r >= 0 && r < M
            && c >=0 && c < N
            && !visit[r][c]);
}
```

ผลลัพธ์

3 4

```
Recursive DFS
(0,0) (0,1) (0,2) (0,3) (1,3) (2,3) (2,2) (1,2) (1,1) (2,1) (2,0) (1,0)
```

นอกจากแบบ recursive แล้ว DFS ยังสามารถเขียนแบบ iterative ได้ โดยการนำ stack เข้ามาช่วยทำงาน กล่าวคือเราจะทดแทน stack ของหน่วยความจำที่ถูกใช้งานเมื่อมีการเรียก recursive call ด้วย stack ที่เราสร้างเองในโปรแกรม โปรแกรม 2.5 ใช้ stack จาก STL และเก็บแถวและคอลัมน์ของ cell ด้วย pair จาก STL

โปรแกรมที่ 2.5 การทำงานของขั้นตอนวิธีของ DFS แบบ iterative บนอาร์เรย์ 2 มิติ

```
#include<iostream>
#include<stack>
#define MAX 1005
using namespace std;
// Globals
int M;                  // number of rows
int N;                  // number of columns
bool visit[MAX][MAX];   // keeps tracks of visited cells
stack< pair<int,int> > S; // for iterative method
// Forwards
void iterativeDFS(int r, int c);
bool isValid(int r, int c);


```

```
int main(){
    cin >> M >> N;
    cout << "Iterative DFS\n";
    iterativeDFS(0,0);
    cout << endl;
    return 0;
}
void iterativeDFS(int r, int c){
    // push just visited cell on stack
    // mark cell as visited
    S.push({r,c});
    visit[r][c] = true;

    // iterate until stack is empty
    while(!S.empty()){
        pair<int,int> cell = S.top();
        int x = cell.first;
        int y = cell.second;
        S.pop();
        cout << "(" << x << "," << y << ") ";

        // order of priority
        // up, right, down, left
        int dx[] = {-1,0,1,0};
        int dy[] = {0,1,0,-1};

        for(int i=0; i<4; i++){
            int xi = x + dx[i];
            int yi = y + dy[i];
            if(isValid(xi, yi)){
                S.push({xi,yi});
                visit[xi][yi] = true;
            }
        }
    }
}
bool isValid(int r, int c){
    return (r >= 0 && r < M
            && c >=0 && c < N
            && !visit[r][c]);
}
```

ผลลัพธ์

3 4

```
Iterative DFS

(0,0) (0,1) (0,2) (0,3) (1,3) (2,3) (2,2) (1,2) (1,1) (2,1) (2,0) (1,0)
```